Photometric Redshifts from Observed Images

with Boosted Decision Trees, Random Forests and CNNs

12.03.2021
David Rufer, Sebastián Gil and Qiang Wang (王强)

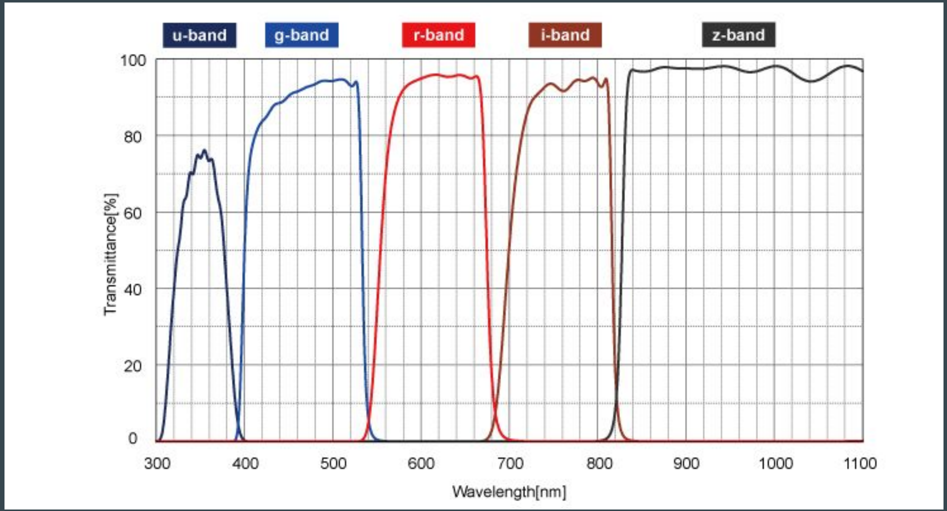
Outline

- Introduction
- Data Preparation
- Machine Learning Algorithms
- Conclusion

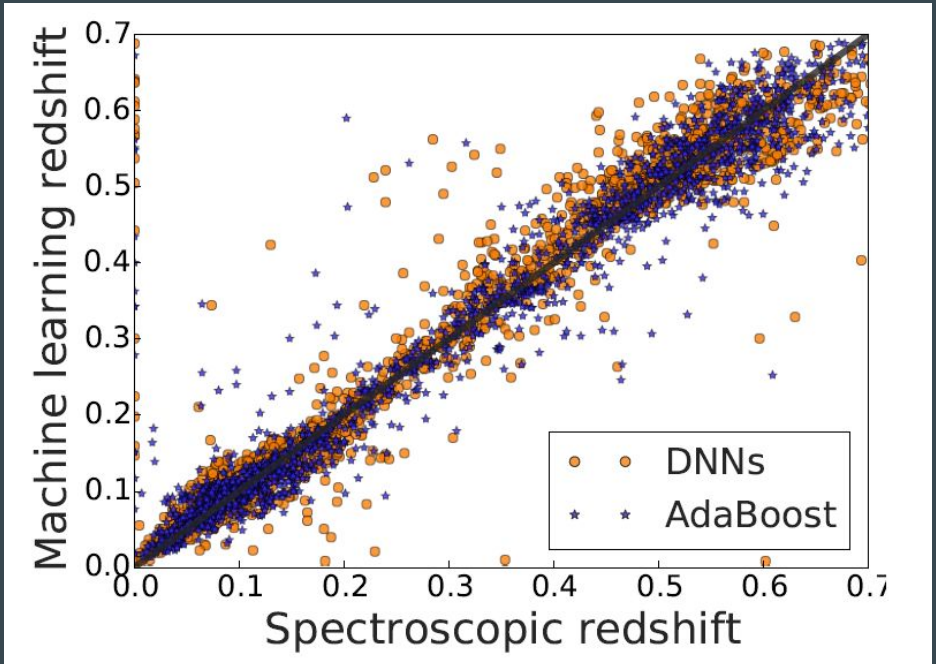
Introduction

Statement of the problem

- Redshifts
- Photometric vs Spectroscopic
- Why use machine learning?
- Why use the whole galaxy image?



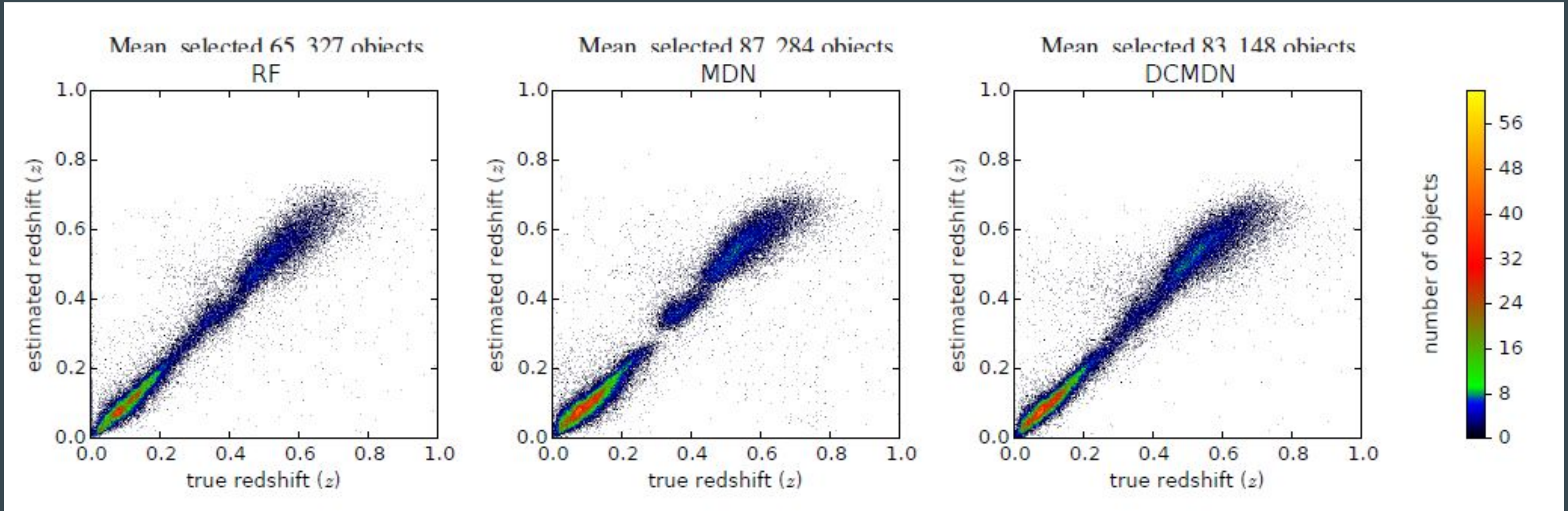
Prior Research



"the choice of which photometric input features to train the machine architecture, from the full list of possible photometric features, is still left to the discretion of the user.... We ***no longer impose our prior beliefs*** *upon which derived photometric features produce the best redshift predictive power....* ***we completely remove the user*** *from the photometric redshift estimation process".*

Hoyle, 2015

Prior Research



rms = 0.0431

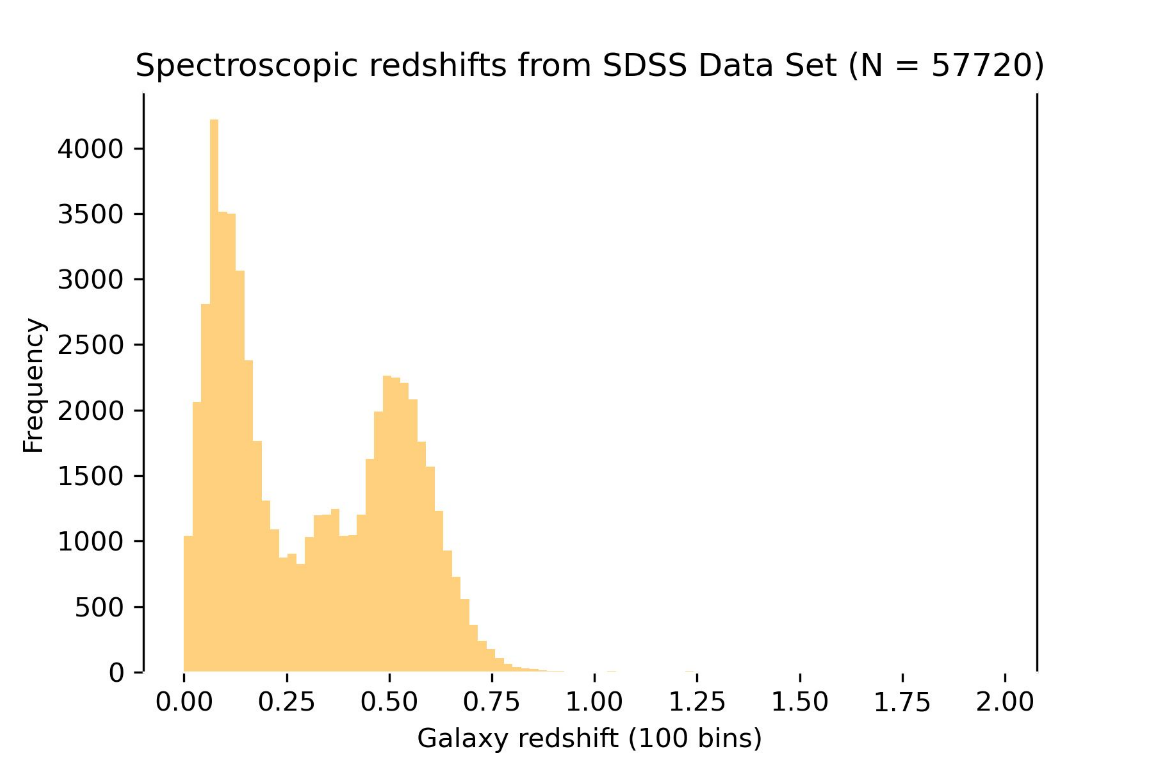
rms = 0.0442

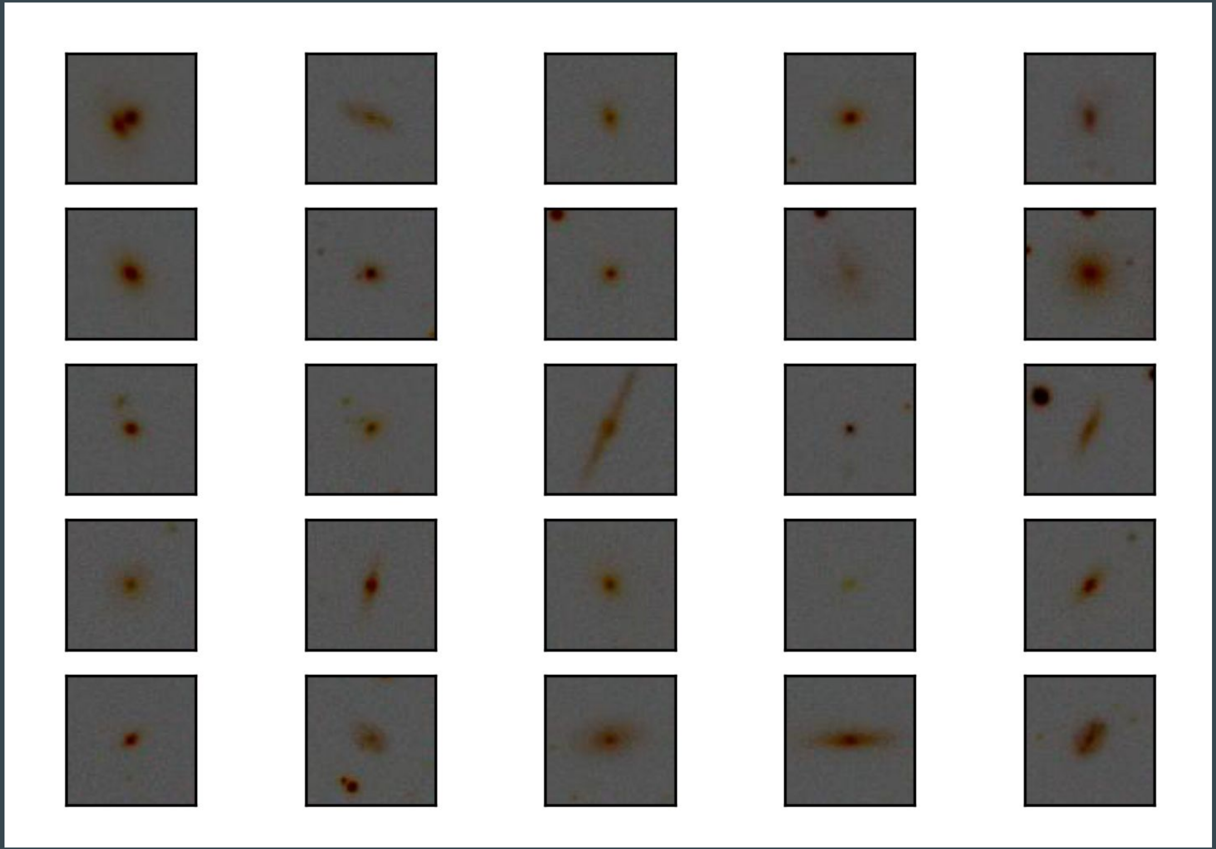
rms = 0.0409

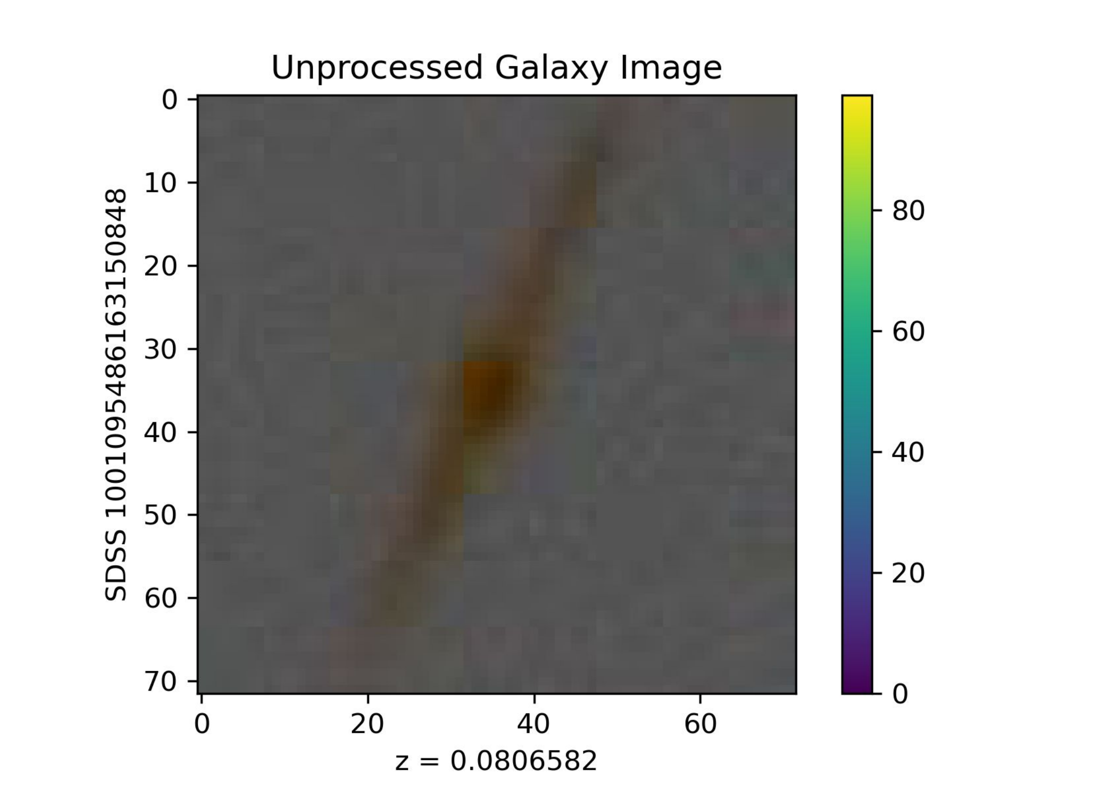
D'isanto and Polsterer, 2019

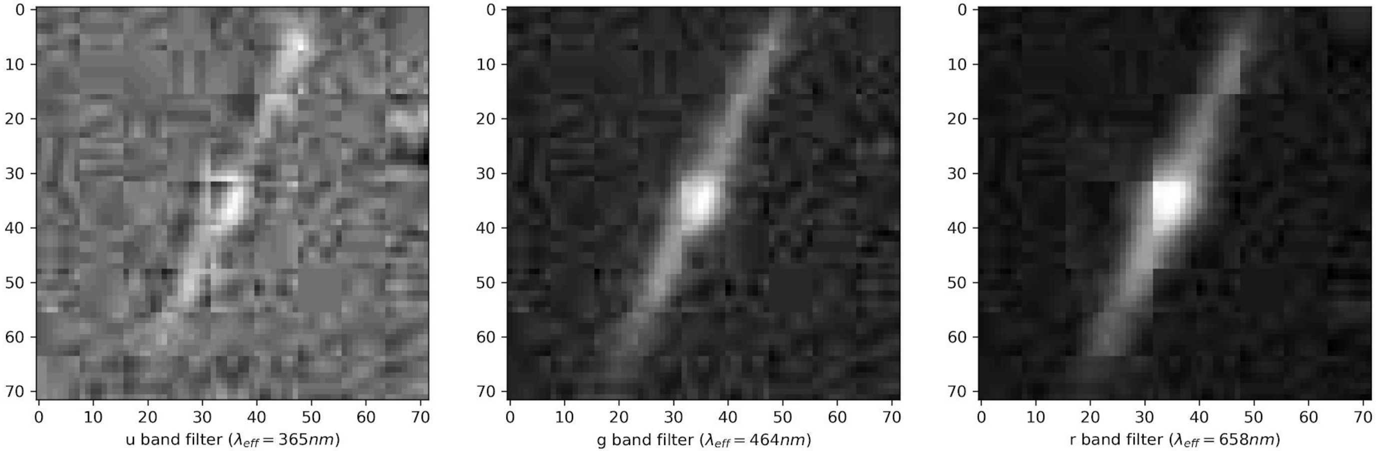
SDSS Dataset

Feature	Value
Number of galaxies	57720
Minimum redshift	1.47×10^{-7}
Maximum redshift	1.98
Average redshift	0.31
Median redshift	0.28
Mode of redshifts	0.13





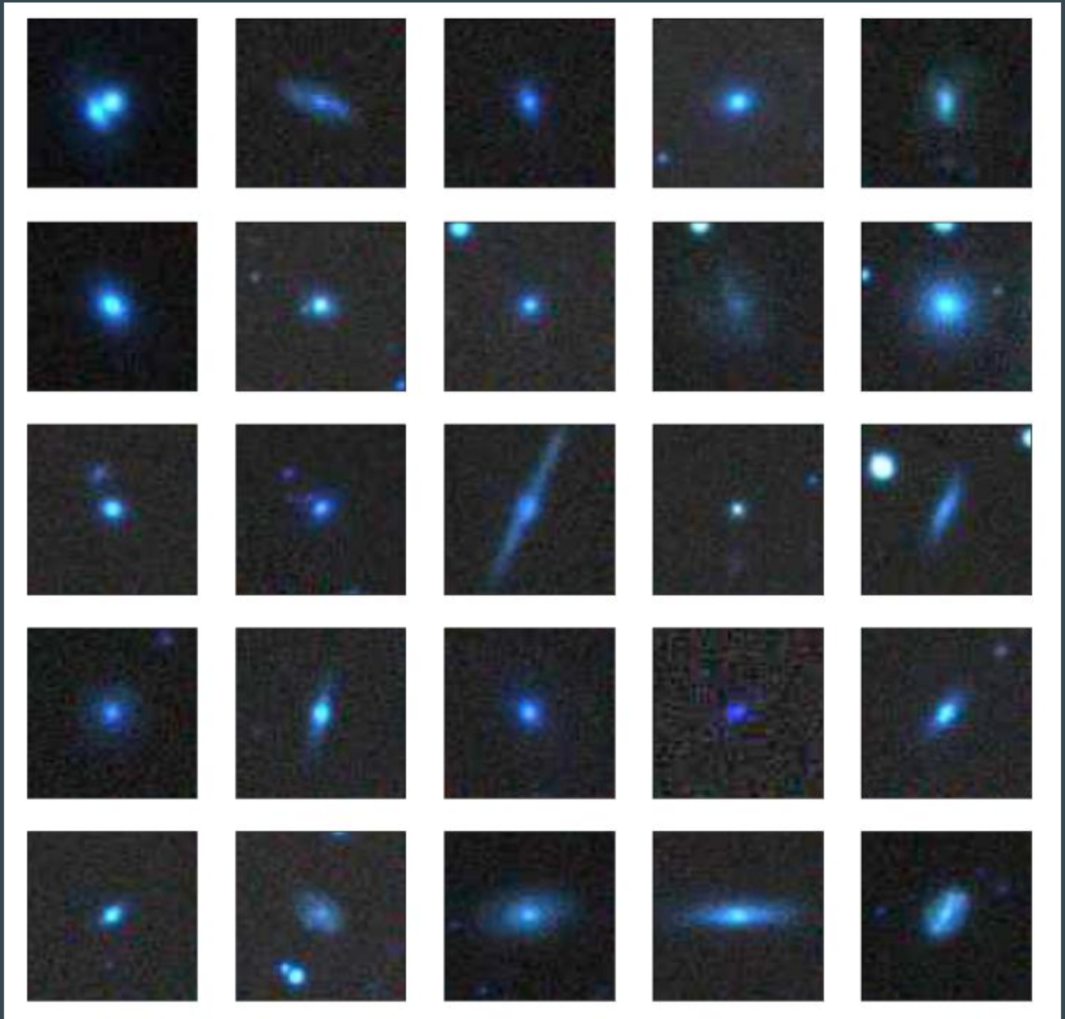

Unprocessed Galaxy Image
SDSS 1001095486163150848
z = 0.0806582
0
10
20
30
40
50
60
70
0
20
40
60
80
60
40
20
0


0
10
20
30
40
50
60
70
u band filter ($\lambda_{eff} = 365nm$)
g band filter ($\lambda_{eff} = 464nm$)
r band filter ($\lambda_{eff} = 658nm$)

Data Preparation

Structure

- Import Data and Redshifts
  - Flipping and Rotating
- Rescaling
  - Inverting
- Cropping
  - Random vs. Maximum Crop
- (Flattening)
- Splitting



Algorithms Used

Boosted Decision Trees

What are boosted decision trees?

- Combination of multiple weak learners into a single strong learner

$$F_{m+1}(x) = F_m(x) + h_m(x)$$

$$h_m(x) = -\frac{\partial L_{\mathrm{MSE}}}{\partial F}$$

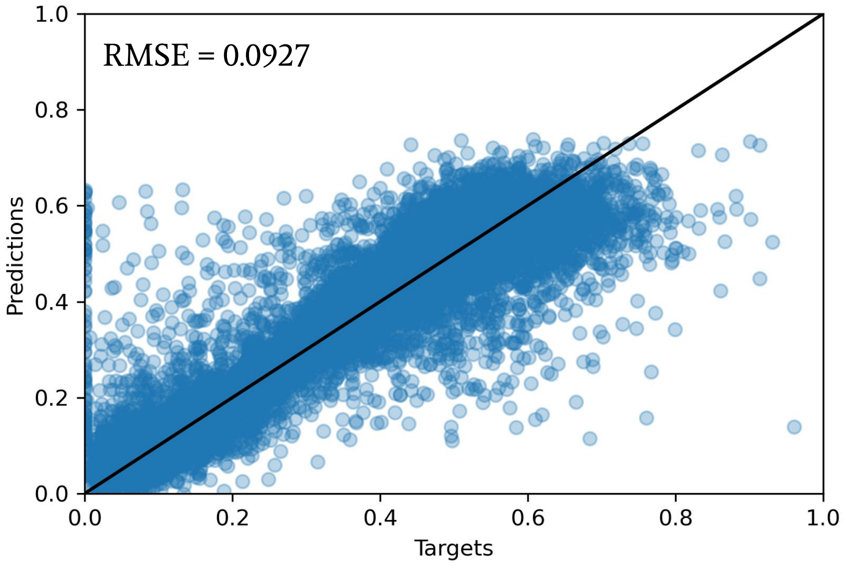
Hyperparameter Search with HyperOpt

```
space ={
    'loss': hp.choice('loss', ['ls', 'huber', 'lad']),
    'criterion': hp.choice('criterion', ['friedman_mse', 'mse']),
    'learning_rate': hp.loguniform('learning_rate', np.log(0.01), np.log(0.4)),
    'n_estimators': hp.randint('n_estimators', 150),
    'min_samples_split': hp.randint('min_samples_split', 50),
    'min_samples_leaf': hp.randint('min_samples_leaf', 51),
    'max_depth': hp.randint('max_depth', 16)
}
```

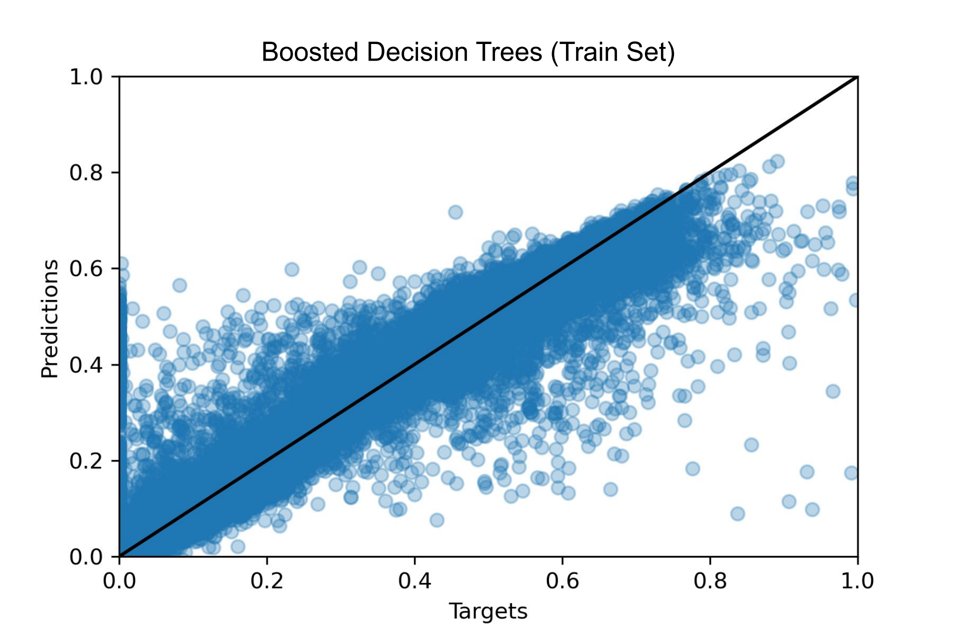
```
GradientBoostingRegressor(learning_rate=0.52, loss='huber', max_depth=14,
                          min_samples_leaf=12, min_samples_split=43,
                          n_estimators=77)
```

Results (change titles)

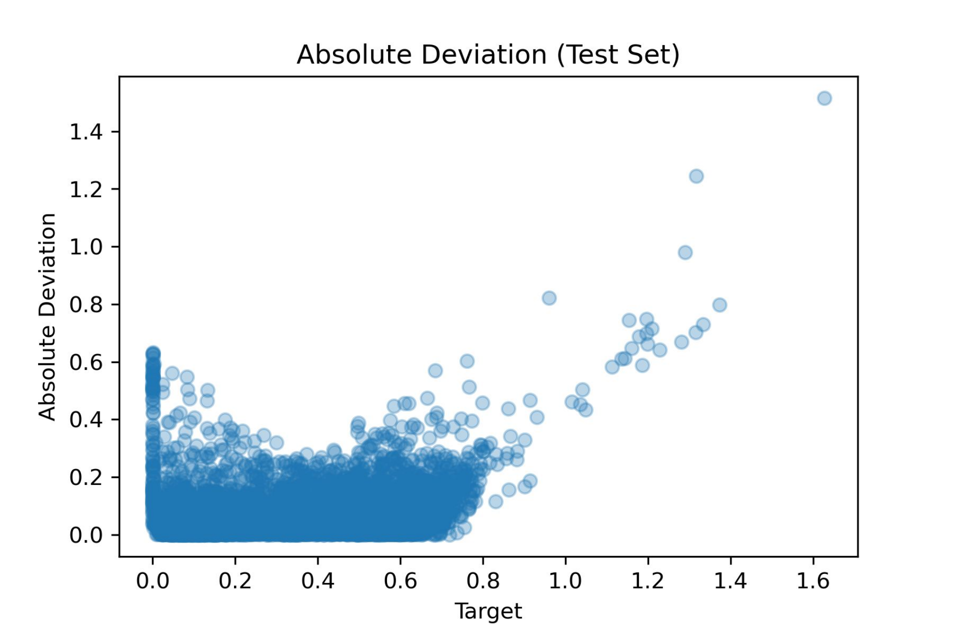
```
GradientBoostingRegressor(learning_rate=0.52, loss='huber', max_depth=14,
                          min_samples_leaf=180, min_samples_split=650,
                          n_estimators=77)
```



Results

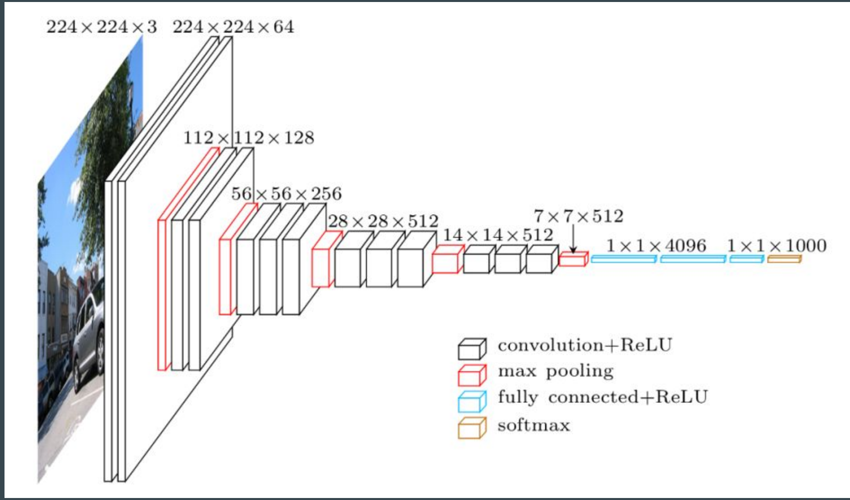

Boosted Decision Trees (Train Set)
Predictions
Targets
0.0
0.2
0.4
0.6
0.8
1.0

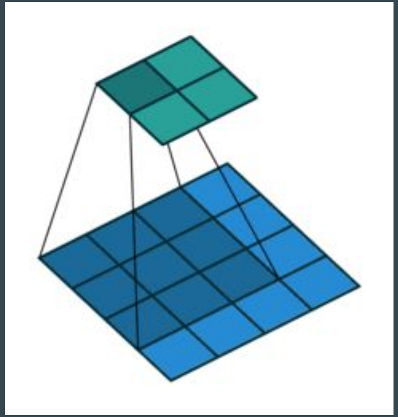
Results


Absolute Deviation (Test Set)
Absolute Deviation
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
Target

Convolutional Neural Network

Review of CNNs





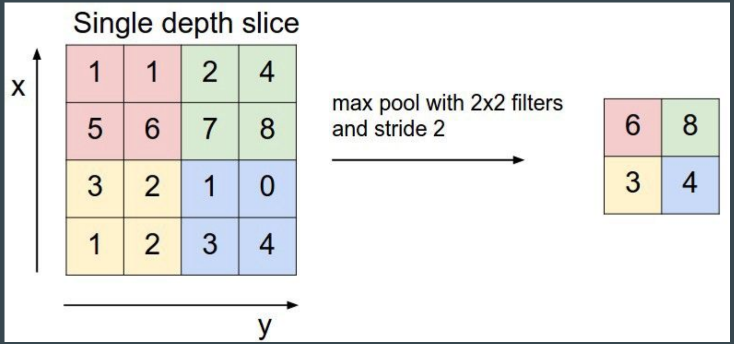
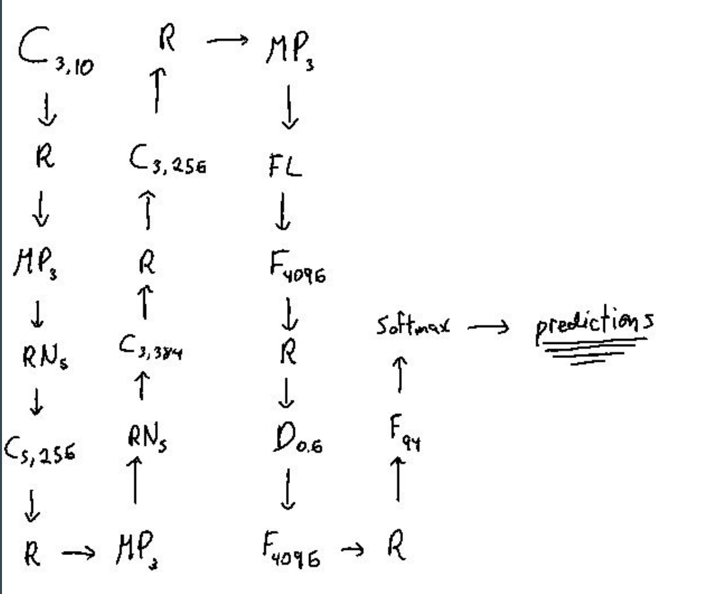


Image credit:
towardsdatascience.com

Classification vs. Regression Tasks

	Classification	Regression
targets	class list of features	array of values
loss	cross-entropy	mse, mae, logcosh
input/output	one hot output layer and labels	single output
final activation	softmax	linear

Architectures Considered



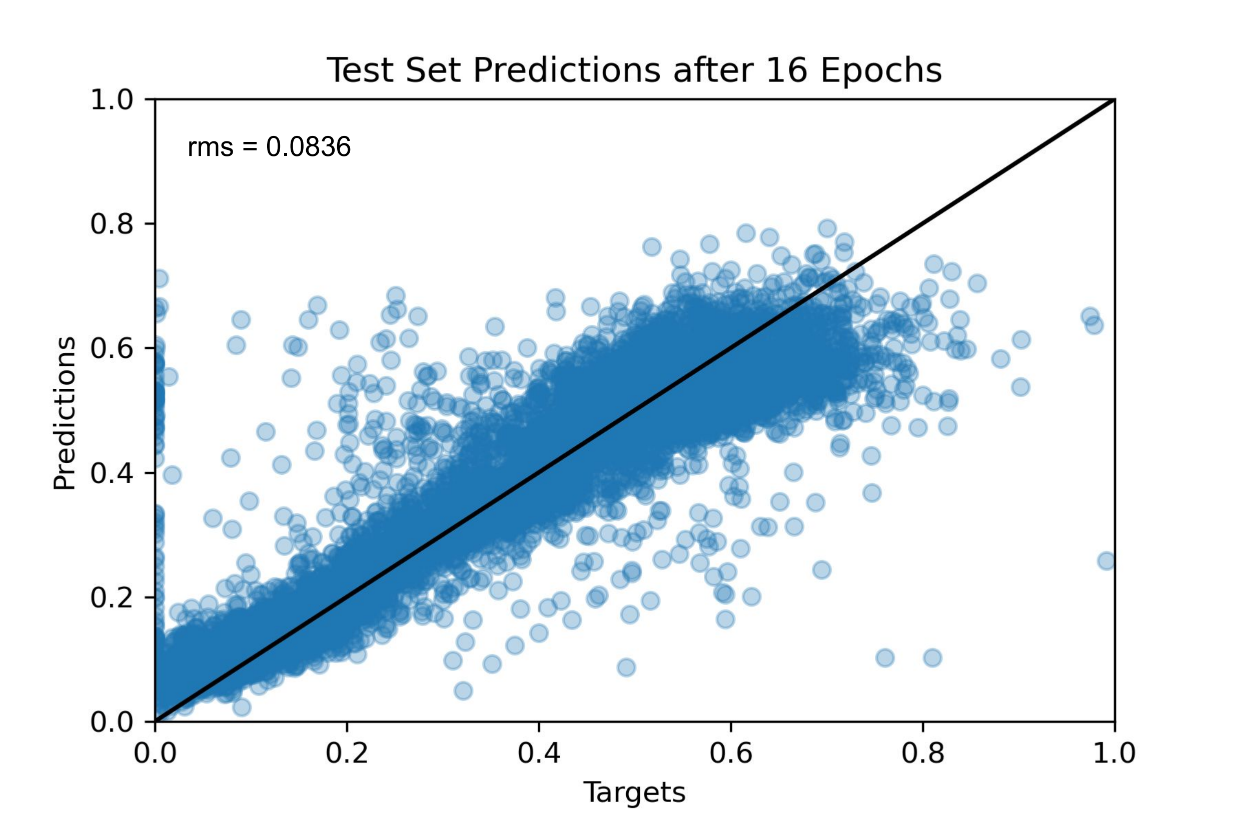
- Implemented in GraphLab (deprecated)
- Little reference —> hard to implement
- Poor performance in Keras
- Replicability...

Architectures Considered

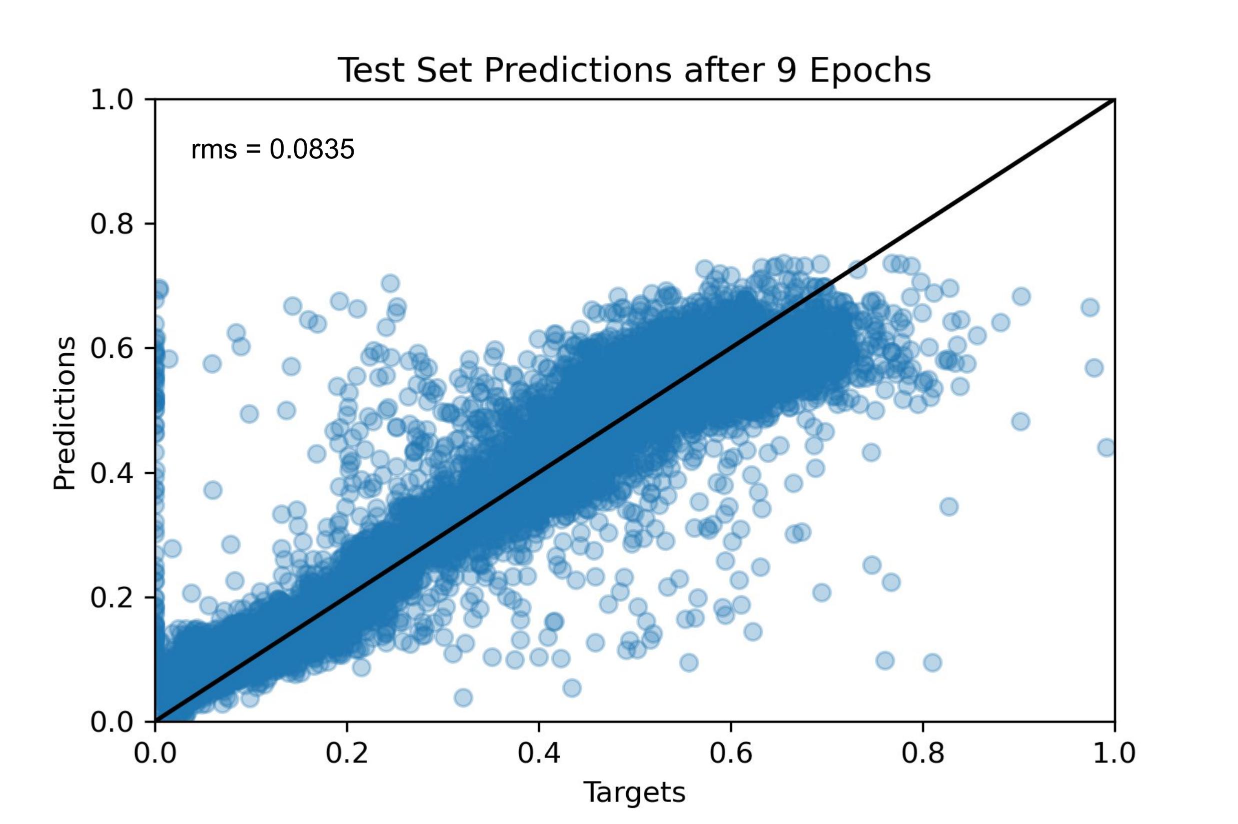
```python
def createCNNModel():
    model = Sequential()
    model.add(Conv2D(10, (5, 5), padding='same', input_shape=(60,60,3)))
    model.add(BatchNormalization())
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3)))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3)))
    model.add(Flatten())
    model.add(Dense(2000, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(500, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(300, activation='relu'))
    model.add(Dense(30, activation='relu'))
    model.add(Dense(20, activation='relu'))
    model.add(Dense(20, activation='relu'))
    model.add(Dense(1, activation=tf.keras.activations.linear,
     bias_initializer=tf.keras.initializers.Constant(0.5)))
    return model
```

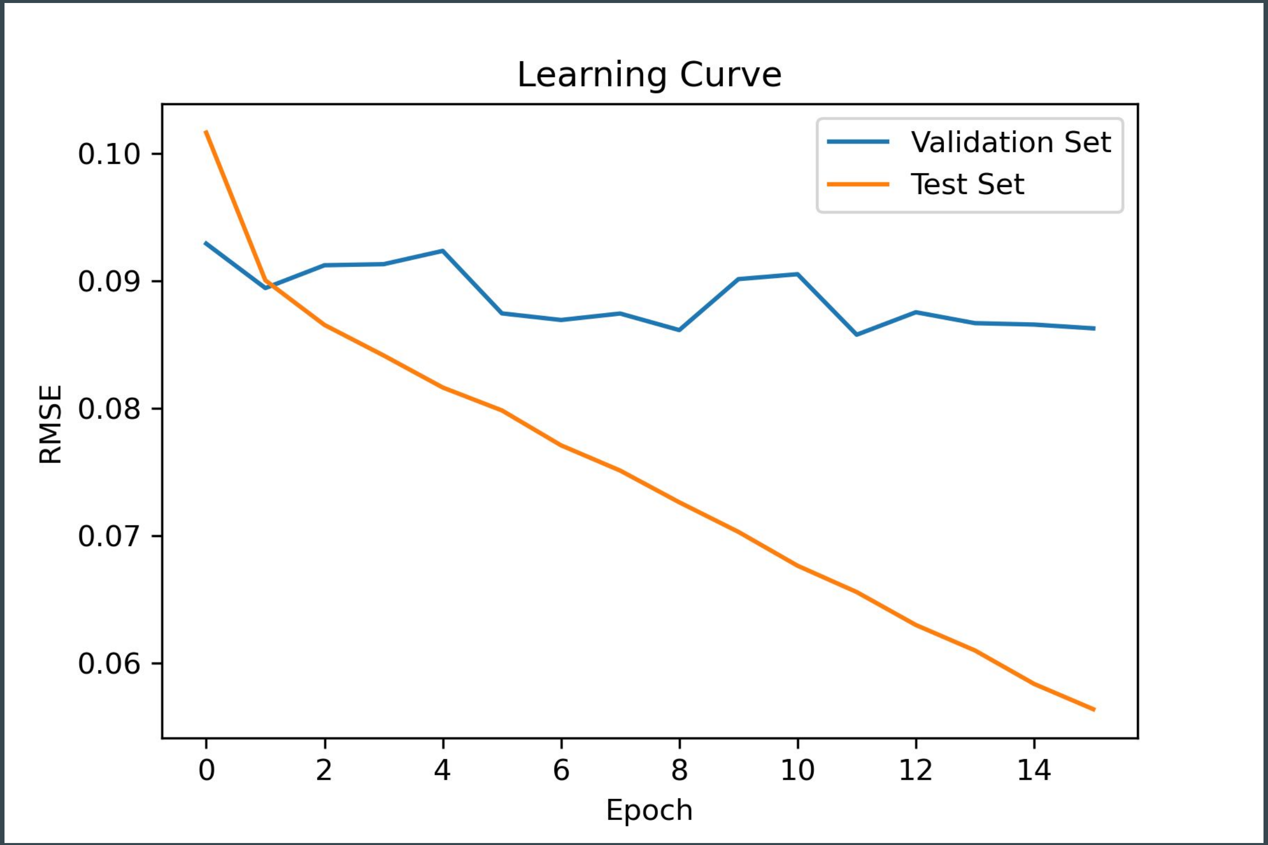
Hyperparameters	Values explored
learning rate	$0.1 - 0.00001$
batch size	$\{32, 64\}$
loss	{'mae', 'mse', logcosh'}
epochs	$\{4 - 16\}$
dropout	$\{0.2, 0.4, 0.6\}$

Results



Test Set Predictions after 16 Epochs
rms = 0.0836
Predictions
Targets
0.0
0.2
0.4
0.6
0.8
1.0


Test Set Predictions after 9 Epochs
rms = 0.0835
Predictions
Targets
0.0
0.2
0.4
0.6
0.8
1.0


Learning Curve
Validation Set
Test Set
RMSE
0.10
0.09
0.08
0.07
0.06
0
2
4
6
8
10
12
14
Epoch

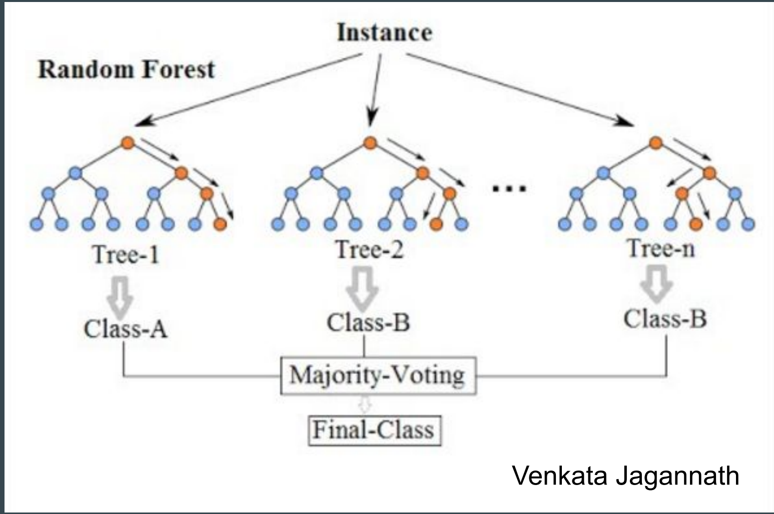
Random Forests

Random Forest Basics

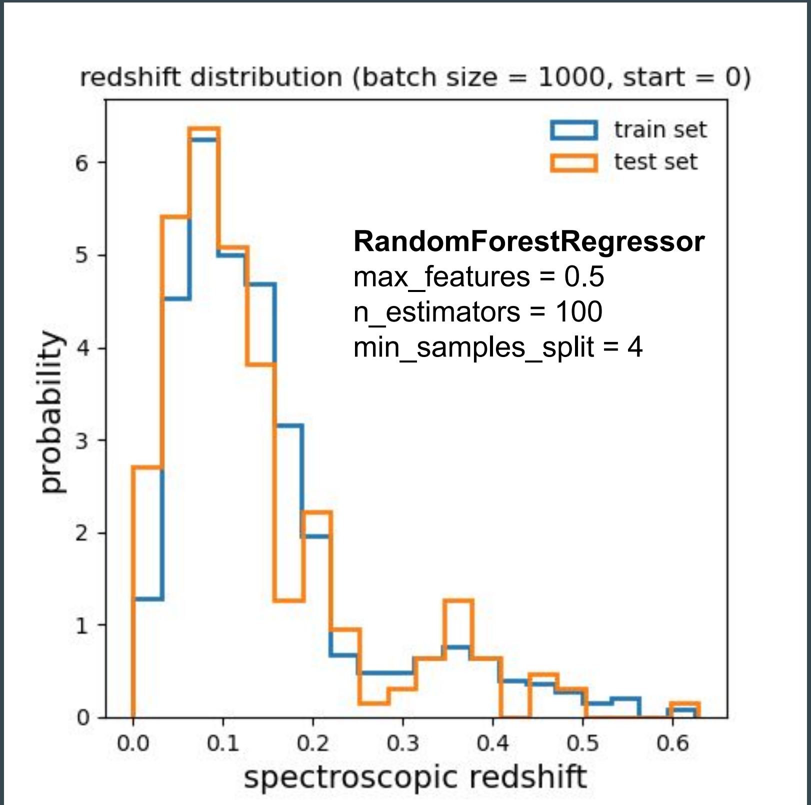
RF, aka random decision forest, is based on decision tree method and improves it by constructing a multitude of decision trees at training time.

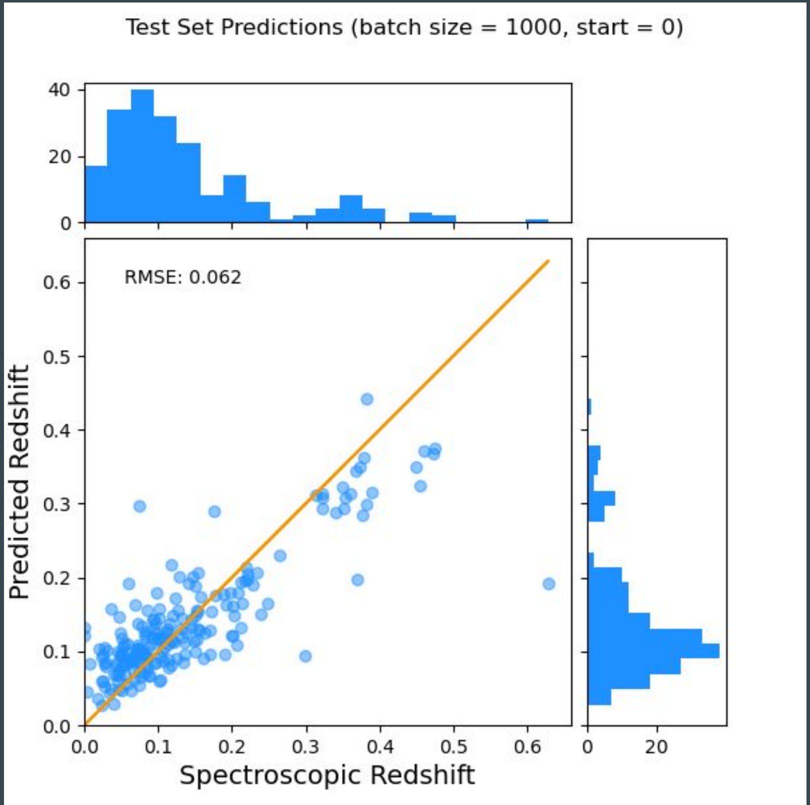
The output is the average of the individual tree, for regression.

This method overcomes the overfitting problem with decision tree.



Random Forest: Quick Example





RandomForestRegressor: Hyperparameter tuning

- **n_estimators**: number of trees, the larger the better
- **max_features**: size of the random subsets of features to consider when splitting a node (our data has 10800 features), options: none, sqrt, float, int etc.
-
- **bootstrap** (True/False): if bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.
- **max_depth**: The maximum depth of the tree
- **min_samples_split**: the minimum number of samples required to split an internal node
- **oob_score**(True or False): whether to use out-of-bag samples to estimate the R^2 on unseen data.
- ...

Random Forest: Hyperparameter tuning with GridSearchCV

n_estimators = np.arange(100, 300, 2)

max_features = np.concatenate(([None, "sqrt", "log2",], np.linspace(0.1, 0.99, 5)))

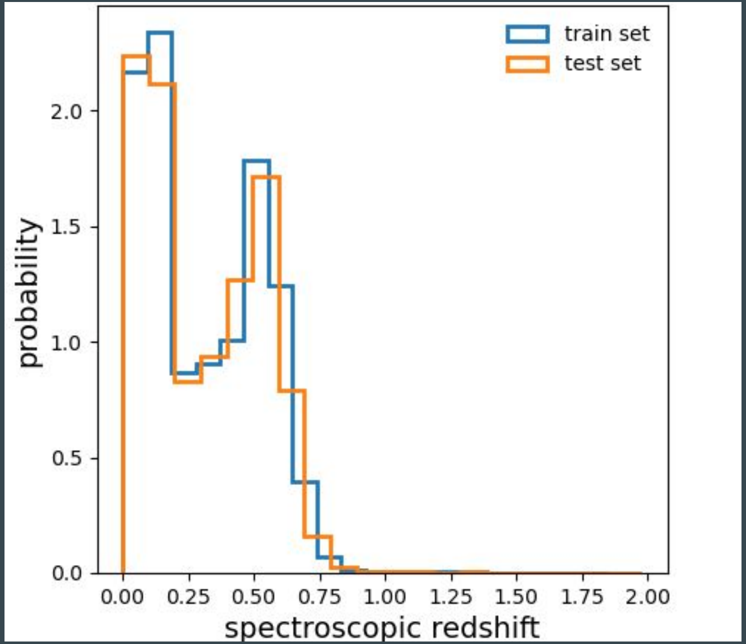
bootstrap = [True, False]

oob_score = [True, False]

min_samples_split = [2, 4]

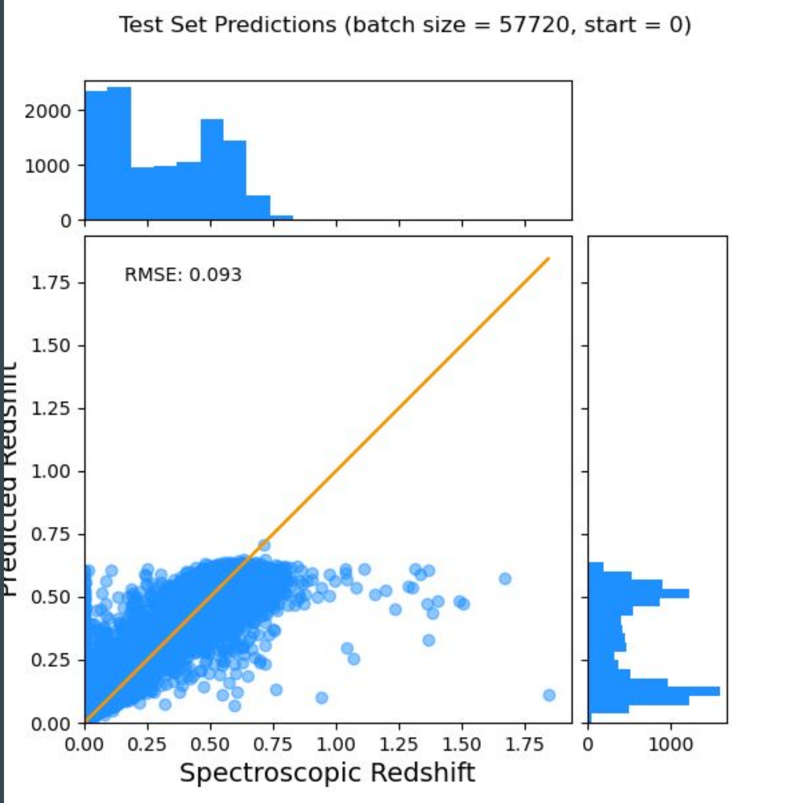
max_depth = [None]

cv=5, scoring='neg_mean_squared_error'

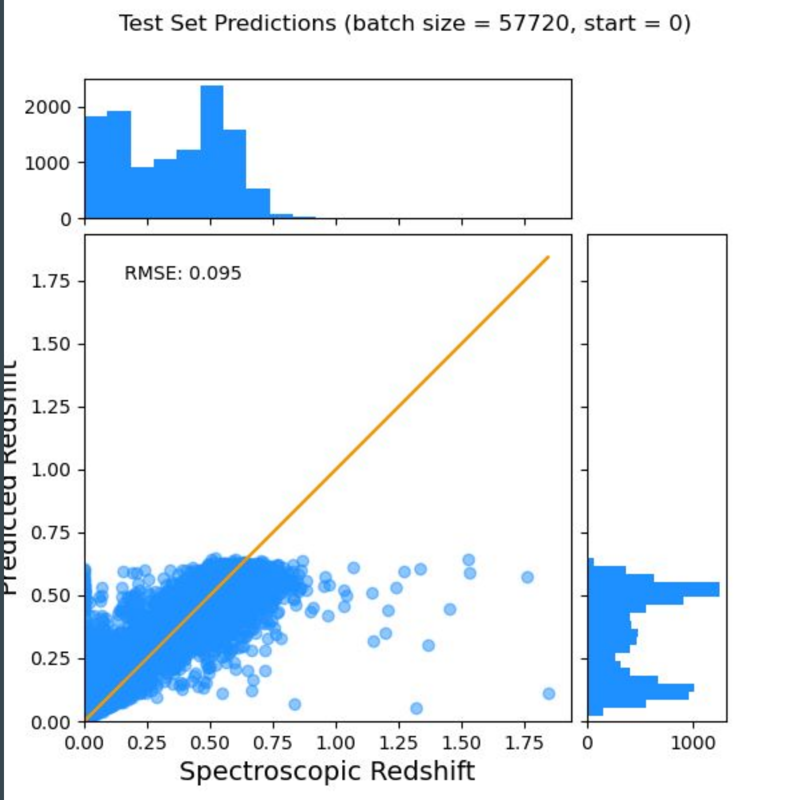


Random Forest: Results

```
bootstrap: True
max_depth: None
max_features: None
min_samples_split: 2
n_estimators: 194
```




```
'bootstrap': False,
'max_depth': None,
'max_features': sqrt,
'min_samples_split': 2,
'n_estimators': 190
```



Conclusion

Comparison of Results

Methods	RMSE
Boosted Decision Trees	0.0927
Convolutional Neural Network	0.0836
Random Forests	0.095

D'isanto and Polsterer, 2019 gives RMSE ~ 0.04

Moving Forward

- Coding best practices
  - test-driven development
  - shared repository
  - version control
  - class-based methods
  - memory optimization
- Replicability of code for robustness analysis
  - under parameter changes
  - under change in idealizing assumptions

References

- Astrophysical
  - *Hoyle, Measuring photometric redshifts using galaxy images and Deep Neural Networks*. arXiv, 2015.
  - D'Isanto and Polsterer. *Photometric redshift estimation via deep learning*. A&A 609, A111, 2018.
  - Pasquet, Bertin, et al. *Photometric redshifts from SDSS images using a Convolutional Neural Network*. A&A 621, A26, 2019.
- Machine Learning
  - Documentation and tutorials for Keras, Tensorflow, etc.
  - Towards Data Science, multiple blogs, etc.